ECOS: Practical Mobile Application Offloading for Enterprises

Aaron Gember, Charlotte Dragga, Aditya Akella
University of Wisconsin, Madison
{agember,dragga,akella}@cs.wisc.edu

Abstract

Offloading has emerged as a promising idea to allow handheld devices to access intensive applications without performance or energy costs. This could be particularly useful for enterprises seeking to run line-of-business applications on handhelds. However, we must address two practical roadblocks in order to make offloading amenable for enterprises: (i) ensuring data privacy and the use of trusted offloading resources, and (ii) accommodating offload at scale with diverse handheld objectives and compute resource capabilities. We present the design and implementation of an Enterprise-Centric Offloading System (ECOS) which augments prior offloading proposals to address these issues. ECOS uses a logically central controller to opportunistically leverage diverse compute resources, while tightly controlling where specific applications offload depending on privacy, performance, and energy constraints of users and applications. A wide range of experiments using a real prototype establish the effectiveness of our approach.

1 Introduction

Handheld devices such as smartphones and tablets are being increasingly recognized as critical business tools, and enterprises are targeting both specialized (e.g., line-of-business) and common (e.g., dictation and transcription) mobile applications to these platforms [15]. Unfortunately, the complexity and overhead of the applications [10, 12], and the accompanying data protection issues, are seen as major impediments to full-fledged deployment on handheld devices [2, 4].

Application-indepedent offloading frameworks have long been recognized as an important mechanism for enabling smartphone users to access resource-intensive applications without incurring energy and performance costs [7, 9, 10, 14, 16]. As handhelds become the primary platforms for some employees, we believe mobile application offloading will be essential for running resource-intensive enterprise applications–e.g., modeling and analysis tools, handwriting and speech recognition, etc.—with suitable performance and energy usage. However, two key roadblocks currently prevent enterprise adoption of mobile application offloading.

1. Privacy and trust: Enterprise applications frequently operate on data with strict privacy requirements, requiring the use of trusted resources (e.g., servers in an enterprise data center) for application execution. The majority of offloading systems ignore such privacy requirements, selecting compute resources solely based on connectivity characteristics and processing capabilities [8]. Even systems which are capable of limiting execution to specific compute resources [7] are insufficient, as they overly restrict offloading opportunities and may unnecessarily impose energy and latency costs.

2. Resource sharing and churn: Enterprises may have thousands of employees using handheld devices, all of which may desire offloading simultaneously. While existing systems address *what* and *how* to offload from a single device [10, 14], no attention has been given to the effects of multiple devices with different objectives simultaneously offloading to the same compute resources. The energy and latency benefits of offloading assumed by some frameworks to be fixed [8, 10] will, at enterprise scale, be quite dynamic. This dynamism increases even more when considering the range of potential compute resources at the disposal of enterprise users—idle desktops, dedicated servers, remote clouds—and the changes in capacity and availability that accompany this diversity.

We present an Enterprise Centric Offloading System (ECOS) which can be coupled with existing offloading frameworks to address the above roadblocks, enabling handhelds to play a significant role in enterprises. ECOS's central design guideline is to allow *many mobile application offloads to opportunistically leverage diverse compute resources, while tightly controlling where specific applications offload depending on privacy, performance, and energy constraints of users and applications.*

ECOS leverages an enterprise-wide controller to orchestrate all application offloads. The controller relies on simple, yet expressive, privacy levels for administrators to encode the data privacy requirements of applications and users. These privacy levels are translated to (*i*) directives to mobile devices, when necessary, to encrypt offloading state transfers, and (*ii*) limitations on the compute resources considered as offloading destinations. Applying encryption only when necessary and considering as many potential compute resources as possible ensures data privacy requirements are met while minimizing overhead and maximizing offloading opportunities. Furthermore, the controller uses fine-grained compute resource management to multiplex offloads from many devices onto a diverse set of resources. Careful resource assignment guarantees offloading provides the desired benefit in terms of latency, energy or both.

Key challenges arise in designing ECOS. First, we

show that securing offloads adds non-trivial latency and energy overhead. Hence, careful choices must be made in deciding whether to encrypt communications and whether specific compute resources should be used. Second, because a limited set of diverse compute resources are shared by a variety of mobile applications with differing performance, energy and privacy requirements, we must design clever allocation algorithms that adapt to diverse application demands and changing resource availability and ensure handhelds see equitable and substantial benefits. Third, our approach should minimize the amount of work handhelds undertake and shift a majority of the decision-making to the controller. We describe our solutions in Sections 3 and 4.

We evaluate a prototype of ECOS we developed for the Android platform [5]. Using two applications representative of enterprise workloads, 12 smartphones and up to 6 servers, we measure the benefits ECOS can provide in a small enterprise setting where devices have varying goals and privacy constraints. Performance improves by as much as 94% and energy savings can be up to 47%.

2 Prior Offloading Proposals

In offloading, parts of a mobile application are run on a remote compute resource to improve performance, lower energy usage, and/or offer higher utility. Many offloading systems have been developed in the past decade. AIDE dynamically partitions memory-demanding mobile Java applications, minimizing the required communication between the handheld and the compute resource [14]. Chroma uses developer-specified execution strategies (i.e., tactics) to divide execution of code modules with varying complexity and accuracy between local and remote resources [7, 13]. MAUI offloads methods from .NET applications to a remote runtime environment based on a history of energy consumption [10]. CloneCloud uses function inputs and an offline model of runtime costs to dynamically partition applications between a weak device and the cloud, with the goal of increasing performance or improving failure resiliency [8, 9].

None of the proposals directly addresses the privacy requirements of offloaded applications. Chroma provides some notion of resource trust [7], but with limited flexibility (Section 3). Alternative methods of augmenting a mobile device's capabilities require the use of specialized APIs [17, 19] or complex trust establishment schemes [16]. ECOS addresses privacy through explicit identification of data privacy levels required by applications and users. These privacy levels are used to make dynamic decisions about whether to encrypt offloading communication and which resources to consider using.

Furthermore, existing proposals focus on *what* and *how* to offload from a single mobile device and do not consider the effects of multiple offloads sharing compute resources. In contrast, ECOS uses a central controller to orchestrate all offloads, thus enabling careful resource assignment to ensure many devices with varying goals (energy savings, latency improvements, etc.) are able to realize benefits from offloading. ECOS also takes advantage of the ability to reduce offloading overhead by caching execution state on a compute resource [10], but it does so in a way that considers resource churn.

3 Privacy and Trust

Offloading introduces the possibility for data to leave the confines of a mobile device without an application's explicit actions. Thus, data privacy, which is paramount in enterprises, becomes an important concern.

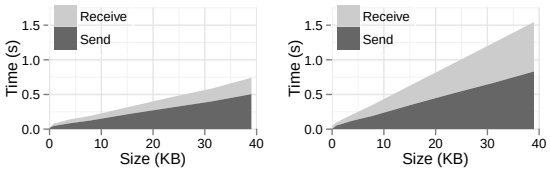
The challenge in ensuring data privacy is balancing the privacy requirements of enterprises with the ability to offer significant energy and performance benefits from offloading. Sufficient privacy protection means both (*a*) protecting execution state in transit and (*b*) limiting the compute resources used for offloaded execution. One way to address (*b*) is to statically specify the compute resources each application trusts [7]. However, this eliminates the ability to adjust compute resource selection based on the privacy requirements of the current application invocation: e.g., a mobile user may utilize dictation software to create confidential legal documents, or they may use the application to write non-sensitive emails. Flexibility is also important for addressing (*a*), as the high overhead of encrypted communication (Section 3.1) can significantly reduce offloading benefits.

We seek to identify privacy requirements at a sufficiently fine granularity for offloading to be both *secure* and *beneficial*. ECOS can rely on an admin-specified policy to identify privacy requirements for specific applications/users, or ECOS can leverage information flow tracking [11] for more dynamic decisions (Section 3.2).

3.1 Security Overhead

The simplest way to ensure privacy is to apply strictest-case security mechanisms to all offloads regardless of the data involved, e.g., encrypt all offloading communication. This avoids the complexity and overhead of identifying the privacy requirements of an offload, but it can decrease offloading benefits and opportunities.

We illustrate the potential impact by quantifying the overhead of always securing execution state in transit. Our measurements use an Android emulator [5] and a 2GHz dual-core desktop running a native x86 version of Android in a virtual machine [6]. The emulator and VM both run our prototype capable of capturing and loading execution state. We offload a single method call which requires transferring a specific amount of execution state.



(a) Unencrypted (b) Encrypted

Figure 1: End-to-end latency overhead

Figure 1 shows the latency overhead of offloading both (a) without and (b) with TLS encryption. The latency overhead more than doubles when encrypting execution state in transit, translating to a direct reduction in the performance benefits possible with offloading. For example, if we transfer 15KB of execution state *unencrypted*, execution must be at least *0.3s faster* on the compute resource, versus the mobile device, to provide performance benefits. Whereas, if we transfer the same 15KB of state *encrypted*, execution must be at least *0.8s faster* to realize any performance benefits. Furthermore, overhead increases with the size of the transferred state. We discuss in Section 4.2 how to counter this effect by preferring the same compute resource for sequential offloads and caching execution state on the resource. We also observe increases in energy overhead with encryption and with larger state sizes but exclude the results for brevity.

The high overhead of encryption illustrates the importance of applying privacy mechanisms *only when needed*.

3.2 When to Secure

Identifying when to secure in order to ensure privacy requires determining the nature of data/state involved in offloading. The data involved in a specific application offload can be *enterprise-private*, *user-private*, or *not-private*. *Enterprise-private* data should not be leaked outside the enterprise, e.g., intellectual property such as internal procedure documents. *User-private* data should only be accessed by the current user of an application, e.g., a patient's medical record can only be accessed by individuals working with the patient, per federal law.

In our ECOS prototype, we determine the privacy level of an offload based on privacy levels statically assigned to mobile devices and applications by an administrator. Mobile devices may be classified based on who they belong to, e.g., the CEO's smartphone is user-private or all company-owned devices are enterprise-private. Applications are assigned privacy levels based on the strictest level of private data they are expected to access (or generate). Thus, the assigned privacy levels subsume privacy constraints on individual data items and bytes. An application is classified as not-private only if it has *zero* likelihood of accessing private data. Existing enterprise security policies, e.g., access control lists, or developer input may aid admins in assigning these privacy levels.

We acknowledge that the granularity of applications and users is quite coarse: ideally, an application should be assigned a privacy level based on the specific data objects it is accessing at a given point in time. To do this, we propose extending ECOS with mechanisms to track the flow of private information on the mobile device [11]. Data with known privacy requirements—e.g., data coming from specific servers, email senders, or file locations—can be tracked, enabling dynamic knowledge of the data contained in an offload. Given details on which types of data are involved in an offload, either the mobile user or our controller can decide which of the three privacy levels applies. Flow tracking has relatively low overhead—TaintDroid only imposes 14% overhead on a CPU-intensive task [11]. The main challenge is deciding what types of data should be tracked and at what granularity, e.g., should data from emails be identified by email sender, by sender domain, or solely by the fact that it came from an email. We leave this as future work.

3.3 Securing Offload

The ECOS central controller uses the privacy level specification to determine what mechanisms are required to ensure sufficient data privacy. The privacy required by the offload is the stricter of the privacy level of the application or the mobile device. The privacy level determines (*a*) whether execution state should be encrypted in transit and (*b*) which compute resources can be used.

A TLS-encrypted connection is established between the mobile device and compute resource when the offload is *user-private*, or when the offload is *enterprise-private* but the handheld or compute resource is outside the enterprise. All other offloads use unencrypted connections to reduce latency and energy overhead. The controller informs the handheld and resource which connection type to use; the resource only accepts connections of that type.

The compute resources considered for offloading are limited to resources which provide a level of trust at least as high as the offload privacy level. For example, a remote cloud may only be trusted with *not-private* offloads, an enterprise data center server with *enterprise-private* offloads, and one's own desktop with *user-private* offloads. The trust level of specific resources is specified by administrators. Next, we discuss how to select a specific resource from the set of suitably trusted resources.

4 Resource Sharing and Churn

The second roadblock addressed by ECOS is supporting simultaneously offloads from many mobile devices. Existing systems address *what* and *how* to offload from a single device [9, 10, 14] but do not consider how multiplexing offloads from several devices on a single compute resource will impact performance and energy benefits. Enabling many handheld devices to opportunistically leverage the diverse compute resources in enter-

prise networks requires careful consideration of (*a*) the performance constraints of different users and applications and (*b*) the changing availability of resources. A further complication arises from the fact that we must assign resources without knowing a priori when other applications will request specific types of resources. Below, we discuss how ECOS addresses these challenges.

4.1 Multiplexing Offloads

Mobile applications typically offload with a goal of either *energy savings* or *performance improvement*. Offloads seeking energy savings are focused on avoiding battery drain on the mobile device; their goal may be satisfied by any trusted, available compute resource. In contrast, offloads seeking performance improvements are focused on having computation run faster on the offloading resource than the mobile device; compute resources must be carefully selected such that the resource has sufficient idle computation capacity to satisfy this goal. A performance seeking offload is not guaranteed to save device energy, as the energy overhead of state transfer may exceed the energy savings from reduced CPU usage.

When there are always more suitable compute resources (idle desktops, servers, etc.) than mobile applications desiring offloading, we can use a simple approach: an application offload is assigned to a dedicated resource; after offloaded execution completes, a different offload can be assigned to the resource. Dedicating a resource's CPU capacity to a single offload ensures performance seeking offloads achieve some benefit.

Unfortunately, this simple approach is of limited use. First, we expect in the future there will be orders of magnitude more mobile applications desiring offloading. The simple approach causes some applications to be denied resources, forcing them to run on the mobile device. Second, the approach cannot support more complex offloads, e.g., parallel offload of different application parts [9].

When there are limited compute resources (e.g., due to trust constraints, reduced resource availability, or a high number of active mobile applications) we assign multiple offloads to the same resource. ECOS relies on a straightforward scheduling heuristic for pairing offload requests with resources that are likely to offer the desired benefits and guarantees: Performance seeking offloads must be assigned to resources whose unused CPU capacity exceeds the CPU speed of the mobile device, otherwise no resource is assigned. Energy seeking offloads can be assigned to a resource regardless of its CPU capacity, as we are not concerned with the execution time of these offloads. However, to avoid an energy seeking offload degrading the benefits received by a performance seeking offload, energy seeking offloads are assigned to resources only with other energy seeking offloads.

4.2 Resource Churn

A key issue in resource sharing pertains to whether the compute resource used for offload should be changed during the course of an application's execution. Changing means that consecutive offloads can leverage different resources, enabling fine-grained assignment of resources that dynamically adapts to demand and resource availability over time. However, this eliminates the possibility of caching execution state at compute resources and only sending state deltas between subsequent offloads [10], an issue which becomes especially important when using encrypted connections (Section 3.1). ECOS minimizes state transfer overhead by maintaining *resource affinity* as much as possible: that is, ECOS always seeks to offload an application to the same resource every time. This allows ECOS to keep a partially loaded runtime environment on the compute resource to quickly serve future offloads from the same application and device.

Resource affinity relies upon the assigned resource having a constant amount of CPU capacity available during the lifetime of the application, which may not always be true. Other applications/devices which have an affinity to the same resource may decide to offload at the same time, causing an over-subscription of CPU resources. Alternatively, a compute resource may no longer be available, e.g. due to a cloud server instance being shutdown. There are several options for how to proceed: (*i*) Deny future offloads from the mobile device until CPU capacity is available. This only works with resources which remain running and only temporarily have insufficient CPU resources. While simple, it may not adequately leverage alternative idle resources in the network, and it may prevent a device from offloading for a long period. (*ii*) Assign a new compute resource, requiring the device to resend the full execution state. As mentioned before, this could be quite costly in terms of overhead but still allows offloading to occur, even if the benefits are marginal. (*iii*) Assign a new compute resource and have the original resource migrate the state the next time the device offloads. This is ideal from the perspective of the mobile device, since it can still send a state delta at the next offload. However, state migration significantly complicates the system. The best behavior depends on the characteristics of the network—the number of mobile devices desiring offloading, the duration resources remain available, etc. ECOS is flexible enough to be extended with whichever approach is most suitable.

5 ECOS Prototype

We prototype ECOS using (*i*) Android phones with a modified Dalvik runtime environment, (*ii*) servers running a modified Android image in a virtual machine, and (*iii*) a Python-based central controller.

Smartphones run a custom Android image with a Dalvik runtime environment we modified (4500 LOC) to support offloading. We instrumented Dalvik to check at each method invocation if offloading should be triggered.[1] An "offload agent" running on the phone requests a compute resource from the controller; application execution proceeds directly on the device if no resources are available. Otherwise, Dalvik establishes a connection to the compute resource, using encryption if dictated by the controller, and serializes and sends all method arguments and any needed objects (determined using a mark and sweep algorithm) using custom state serialization code, similar to state transfer in CloneCloud [9]. When offloaded execution completes, Dalvik integrates the results and any object changes into the execution state on the phone, making note of which objects are still cached on the compute resource and keeping the socket connection open for future offloads.

Compute resources run a native x86 version of Android, with the same Dalvik modifications, in a VirtualBox [6] VM. Running a version of Android built for the same architecture as the compute resource is necessary to realize code speed-up. A "restore agent" within the Android VM is responsible for accepting socket connections from phones, launching new Dalvik instances, and loading the received execution state. After execution completes, the state is serialized and sent to the phone; the Dalvik instance remains for subsequent offloads.

The controller (1000 LOC) orchestrates all offloads. Capacity information is received from compute resources and offload requests are received from phones. Using the provided privacy levels policy file, appropriate resources are assigned and the phones and compute resources are instructed whether to encrypt connections.

6 Evaluation

We evaluate ECOS with a range of experiments to establish the viability of supporting resource-intensive enterprise applications on handhelds. We study how well ECOS (*i*) supports enterprise applications with different latency, energy, and privacy needs, and (*ii*) multiplexes offloads and benefits from resource affinity.

We use two representative mobile applications[2]: chess, which we use as an (admittedly artificial) stand-in for a compute-intensive AI-based enterprise application (e.g., non-linear decision-making [1]), and a speech-to-text transcriber, whose behavior we are forced to emulate due to Android's lack of some crucial audio libraries. The chess game plays against itself for 50 moves, with 10s delays every other move for user "think time." The mock speech recognition application models the state size, CPU usage, and memory usage of an actual speech recognizer [3] for 20 recognitions spaced 10s apart.

[1] Resource-intensive methods are statically specified, but ECOS could be extended to select methods at runtime, like MAUI [10].

[2] Real enterprise mobile applications were unavailable due to licensing and lack of source code.

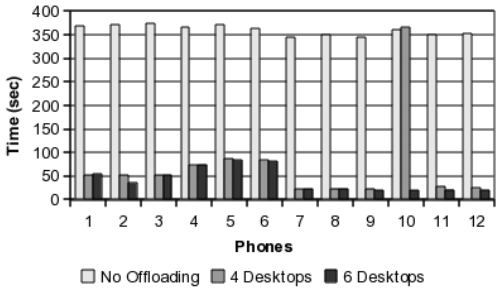


Figure 2: Comparison of application execution times

We measure ECOS for a small enterprise setting using a set of Android emulators and servers. Phone emulators are used because we only had access to a limited number of unlocked Android phones. We confirmed the CPU frequency was similar (20% faster on the emulator), and memory usage was the same, by running both of our applications several times on a real ADP1 phone and in the emulator. Our compute resources are 2.4GHz Intel quad core machines with 4GB of RAM. Our controller runs on a separate machine whose specs are the same. We estimate power consumption using an energy model [18], which takes as input the number of packets and bytes sent/received and CPU usage.

6.1 Full System Analysis

We present an analysis of ECOS for a small enterprise setting consisting of 12 phones (P1-12) and 4-6 servers (S1-6). P1-6 run chess and trust all servers; P7-12 run speech recognition and trust up to 3 servers; P1-3 and P7-9 seek performance improvements while the remaining phones seek energy savings. The constraint of only having 4 servers to serve 12 phones stresses ECOS, but we find it still benefits most applications. With lesser contention when 6 servers are available, the benefits from ECOS become more significant and equitable.

Performance. The execution time (excluding delay between moves/recognitions) for each phone is shown in Figure 2. Without offloading, all applications take approximately 350s to execute. Multiplexing offloads amongst 4 servers significantly reduces the execution time to between 22s and 87s for all phones, including those seeking energy savings, except P10. P10 receives no performance benefit because there is no server it trusts that is available to serve offloads seeking energy savings. Furthermore, P4-6 have 50-60% higher execution times than P1-3 although they are running the same application. This is because P4-6 requested energy savings, while P1-3 requested performance improvement.

Six servers provide enough resources for *all* phones to improve performance. Execution time is 10-20% less than with 4 servers because less multiplexing occurs.

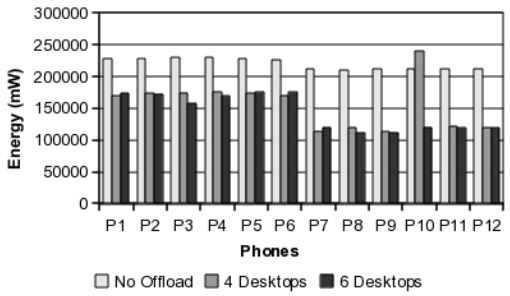


Figure 3: Comparison of application energy usage

Energy. Our experiments show that ECOS also offers energy benefits in the same offloading scenario (Figure 3). The energy savings ranges from 24% to 44% with 4 servers and 23% and 47% with 6 servers. Again with 4 servers, resources are constrained and ECOS is unable to provide benefits to P10. Increasing the available servers allows P10 to attain energy savings equal to its peers.

Although not present here, ECOS may impose energy costs for applications that have requested performance improvements. Likewise, some applications seeking energy savings may see a degradation in performance when there is high resource contention. Since ECOS is opportunistic, the presence of this cost is workload dependent.

6.2 Resource Allocation Efficiency

We now evaluate how well ECOS shares resources amongst many mobile devices. Using the 12 phone, 4 server setup above, we compare a one-to-one resource assignment strategy to multiplexing with and without resource affinity. The execution time for each phone is shown in Figure 4 for the three assignment scenarios.

First, we observe that one-to-one assignment results in less offloading opportunities and higher execution times for two-thirds of the phones. In some cases, e.g., P2, execution takes more than twice as long. This results from the inability to serve more than 4 offloads (as many servers as we have) at any given time. At the same time, a given offload typically takes less time to execute since compute resources are not shared with other offloads: e.g., P10 executes the fastest with one-to-one because it gets full use of the CPU when assigned to a server.

Second, we observe that for most phones, resource assignment with affinity results in lower total execution time. This decrease stems directly from the decrease in state transfer as a result of preserving execution state at the same server, similar to MAUI [10]. However, avoiding affinity can help provide a fair sharer of benefits when the number of compute resources are limited: e.g., P10 receives significant benefit when using multiplexing without affinity as there is more churn in assignments and a greater opportunity for being allocated resources.

In summary, we find that ECOS can support multiple applications with different performance and security needs, offering significant, equitable benefit at low cost.

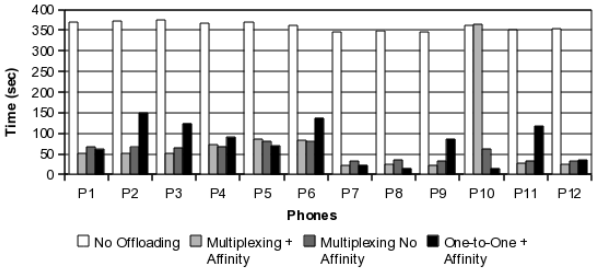


Figure 4: Comparison of different assignment algorithms

7 Conclusion

We presented ECOS, an Enterprise-Centric Offloading System designed to address the security needs of mobile applications and opportunistically leverage available compute resources. ECOS extends the offloading decision process to take into account privacy requirements and costs. In ECOS an enterprise-wide controller assigns trusted compute resources to applications based on resource availability, administrator specified security policies, and the performance or energy savings goals of mobile devices. We showed that ECOS provides both latency and energy benefits, even in the presence of strict privacy requirements and few compute resources, paving the way for wider-spread adoption of offloading to assist current and future enterprise mobile applications.

References

[1] AI enters the mainstream. `http://domain-b.com/infotech/itfeature/20070430_Intelligence.htm`.
[2] Android enterprise security: Mobile phone data protection advice. `http://searchsecurity.techtarget.com`.
[3] CMU sphinx. `http://cmusphinx.sourceforge.net`.
[4] Developing enterprise applications for mobile devices remains way too hard. `http://www.zdnet.com/blog/gardner`.
[5] Google android. `http://android.com`.
[6] Oracle virtualbox. `http://www.virtualbox.org`.
[7] R. K. Balan et al. Tactics-based remote execution for mobile computing. In *MobiSys*, 2003.
[8] B.-G. Chun and P. Maniatis. Dynamically partitioning applications between weak devices and clouds. In *MCS*, 2010.
[9] B.-G. Chun et al. Clonecloud: elastic execution between mobile device and cloud. In *EuroSys*, 2011.
[10] E. Cuervo et al. MAUI: Making Smartphones Last Longer with Code Offload. In *MobiSys*, 2010.
[11] W. Enck et al. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *OSDI*, 2010.
[12] L. Fiering and K. Dulaney. iPads: Not notebook replacements, but still useful for business. *Gartner, Inc.*, 2010.
[13] J. Flinn et al. Balancing performance, energy, and quality in pervasive computing. In *ICDCS*, 2002.
[14] A. Messer et al. Towards a distributed platform for resource-constrained devices. In *ICDCS*, 2002.
[15] S. D. Nelson and D. A. Willis. Separating enterprise tablet applications from consumer apps. *Gartner, Inc.*, 2011.
[16] M. Satyanarayanan et al. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 2009.
[17] S. Smaldone et al. Leveraging smart phones to reduce mobility footprints. In *MobiSys*, 2009.
[18] L. Zhang et al. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *CODES+ISSS*, 2010.
[19] X. Zhang et al. Securing elastic applications on mobile devices for cloud computing. In *Workshop on Cloud computing security*, 2009.